Homework 4 PDS — Answers

iae225 — Ivan Aristy

The schema is: ShoeOrder(brand, styleID, size, color, email, orderDate, basePrice, pricePaid, status, qInStock, qOrdered, bonusPts, phone)

Question 1

Using your group members as customers (and additional fictional data as needed) show an example of a relation on this schema where:

a. Two customers have ordered the same brand/style/color/size of shoes, and

brand	styldeID	size	color	email	orderDate	basePrice
Adidas	Predator	11	blue	iae225@nyu.edu	11-22-24	100
Adidas	Predator	11	blue	iae225@stern.nyu.edu	11-21-24	100

-

pricePaid	status	qInStock	qOrdered	bonusPts	phone
90	inProgress	255	1	0	516-555-5555
90	completed	270	1	0	516-555-5554

b. There are multiple colors and/or sizes of at least one brand/style of shoes, and

brand	styldeID	size	color	email	orderDate	basePrice
Adidas	Predator	11	blue	iae225@nyu.edu	11-20-24	100
Adidas	Predator	10.5	blue	iae225@stern.nyu.edu	11-19-24	100

-

pricePaid	status	qInStock	qOrdered	bonusPts	phone
90	inProgress	255	1	120	516-555-5555
90	completed	270	1	100	516-555-5554

c. There is a customer who has no orders, and

brand	styldeID	size	color	email	orderDate	basePrice
null	null	null	null	iae225@tandon.nyu.edu	null	null

pricePaid	status	qInStock	qOrdered	bonusPts	phone
null	null	null	null	200	516-444-4444

d. There are some black size 9 Adidas Sambas in stock, and no one has ordered any of that shoe type.

brand	styldeID	size	color	email	orderDate	basePrice
Adidas	Samba	9	black	null	null	100

pricePaid	status	qInStock	qOrdered	bonusPts	phone
null	null	100	null	null	null

Comment briefly on problems with this schema that these data illustrate.

There's plenty:

- We must represent a shoe existing with a null or default order, which does not represent a real entity, and, if that were not done, we'd lose track of shoes available if no orders are associated to them
- Integrity and Confusion: Fields like item quantities are hard to keep track of and instantly become outdated when a new order for an item comes in.
- Redundant Data Everywhere. We need unrelated attributes to be defined or null to express a meaningful real entity by itself.
- Lots of for loops would be needed to find relevant data.

Question 2

2. Give an example of a trivial functional dependency in this schema (Recall that a functional dependency alpha α → β is trivial if β ⊆ α) and an example of a functional dependency in which the left-hand side is a superkey. (Such functional dependencies do not violate BCNF)

Trivial: styleID, brand -> brand

Superkey: brand, styleID, size, color, email, orderDate -> basePrice, pricePaid, status, qInStock, qOrdered, bonusPoints, phone

Question 3

3. Write functional dependencies corresponding to each of the following parts of the data description:
   a. Each type of shoe the store carries is identified by its *brand, styleID, size*, and *color*; in addition, the inventory keeps track of the quantity (*qInStock)* of each type of shoe that is currently in stock.

   b. The base price is determined by the brand and styleID. In other words, shoes that have the same brand and styleID always have the same base price, even though they may have different size and/or color.
   c. Each customer has a unique *email,* one *phone number*, and a number of *bonus points*.

1. brand, styleID, size, color -> qInStock
2. brand, styleID -> basePrice
3. email -> phone, bonusPts

Question 4

What is the canonical cover of the set of dependencies implied by the description?

Question 5

Candidate Key:
(brand, styleID, size, color, email, orderDate)

Question 6

A relation is in if for every functional dependency , is a superkey. Let's check:

Consider

Since the candidate key is as above, we miss .

Additionally, A is not trivial since .

Question 7

Decompose ShoeOrder into a collection of schemas each of which is in BCNF. Show your work: at each stage show which schema you are decomposing, which functional dependency violating BCNF you are using for the decomposition, and the resulting two schemas. At the end, show the entire decomposed database schema and give meaningful names to each relation schema.

The schema is: ShoeOrder(brand, styleID, size, color, email, orderDate, basePrice, pricePaid, status, qInStock, qOrdered, bonusPts, phone)

brand, styleID, size, color -> qInStock

- Inventory (brand, styleID, size, color, qInStock)
- R(brand, styleID, size, color, email, orderDate, basePrice, pricePaid, status, qOrdered, bonusPts, phone)

brand, styleID -> basePrice

- ShoePrice (brand, styleID, basePrice)
- R(brand, styleID, size, color, email, orderDate, pricePaid, status, qOrdered, bonusPts, phone)

email -> phone, bonusPts

- User (email, phone, bonusPts)
- R(brand, styleID, size, color, email, orderDate, pricePaid, status, qOrdered)
- Inventory (brand, styleID, size, color, qInStock)
- ShoePrice (brand, styleID, basePrice)
- User (email, phone, bonusPts)
- R: Order (brand, styleID, size, color, email, orderDate, pricePaid, status, qOrdered)

Question 8

Show how the data in (1) is stored using the decomposed database schema from (7). Comment on how the decomposition addresses the anomalies you noted in part (1).

Inventory

brand	styleID	size	color	qInStock
Adidas	Predator	11	blue	255
Adidas	Predator	10.5	blue	270
Adidas	Samba	9	black	100

- Only one entry for the non-differentiated shoe. Q in stock is now kept as an attribute not dependent on orders, allowing us to edit it without having to do something new to orders
- Samba shoes can be kept in DB without depending on null user values or being ordered

ShoePrice

brand	styleID	basePrice
Adidas	Predator	100
Adidas	Samba	100

- basePrice correctly inherits from only brand and styleID, keeping an abstract relationship not dependent on actual shoes in storage

User

email	phone	bonusPts
iae225@nyu.edu	516-555-5555	120
iae225@stern.nyu.edu	516-555-5554	100
iae225@tandon.nyu.edu	516-444-444	200

- 1 was so confusing that I even noticed mistakes I made then, now we can guarantee that users are uniquely identified by their email, and user attributes depend only on those. No need to have null order or inventory values to define a person.

Order

brand	styleID	size	color	email
Adidas	Predator	11	blue	iae225@nyu.edu
Adidas	Predator	10.5	blue	iae225@stern.nyu.edu
Adidas	Predator	11	blue	iae225@nyu.edu
Adidas	Predator	11	blue	iae225@stern.nyu.edu

-

orderDate	pricePaid	status	qOrdered
11-22-24	90	inProgress	1
11-21-24	90	completed	1
11-20-24	90	inProgress	1
11-19-24	90	completed	1

- Orders are uniquely identified, we no longer to make a null order to represent the sambas

Question 9

Suppose the Shoe store changed its price structure to guarantee that all purchases of shoes with the same brand and styleID on the same orderDate have the same PricePaid.

1: Write a functional dependency to characterize this:

1. brand, styleID, orderDate -> pricePaid

2: Decompose your answer from (7) further (if necessary) so it will be in BCNF with respect to the expanded set of functional dependencies
Order (brand, styleID, size, color, email, orderDate, pricePaid, status, qOrdered)

DailyPrice (brand, styleID, orderDate, pricePaid)
Order (brand, styleID, size, color, email, orderDate, status, qOrdered)

3: Is your result dependency preserving?
"

While BCNF ensures robust normalization, it can prevent efficient testing of certain functional dependencies, as shown in a university database example. A ternary relationship `dept_advisor` captures the constraints that instructors belong to one department and students can have at most one advisor per department.

Decomposing this schema into BCNF results in two relations, `(s_ID, i_ID)` and `(i_ID, dept_name)`, but the functional dependency `s_ID, dept_name → i_ID` cannot be enforced without recomputing the join, making the design **not dependency preserving**.

To balance normalization and efficiency, **Third Normal Form (3NF)**, a weaker normal form, allows dependency preservation while maintaining acceptable normalization.
"

Hence, considering our functional dependencies:
brand, styleID, orderDate -> pricePaid
brand, styleID, size, color -> qInStock
brand, styleID -> basePrice
email -> phone, bonusPts

Then, our schema is still in BCNF since we do not have to recompute any joins to guarantee our dependencies check out

Question 10

10. Repeat part 9 with the following supposition (starting with your decomposed database schema from part (7)): Purchases made on the same *orderDate* of shoes with the same *basePrice* have the same *pricePaid*

Inventory (brand, styleID, size, color, qInStock)
ShoePrice (brand, styleID, basePrice)
User (email, phone, bonusPts)
R: Order (brand, styleID, size, color, email, orderDate, pricePaid, status, qOrdered)

1: Write a functional dependency to characterize this:
orderDate, basePrice -> pricePaid

2: Decompose your answer from (7) further (if necessary) so it will be in BCNF with respect to the expanded set of functional dependencies
Order (brand, styleID, size, color, email, orderDate, pricePaid, status, qOrdered)

DailyScalars(basePrice, orderDate, pricePaid)
Order(brand, styleID, size, color, email, orderDate, status, qOrdered)

3: Is your result dependency preserving?
considering our functional dependencies:
orderDate, basePrice -> pricePaid
brand, styleID, size, color -> qInStock
brand, styleID -> basePrice
email -> phone, bonusPts

Then, again, our schema is still in BCNF since we do not have to recompute any joins to guarantee our dependencies check out.